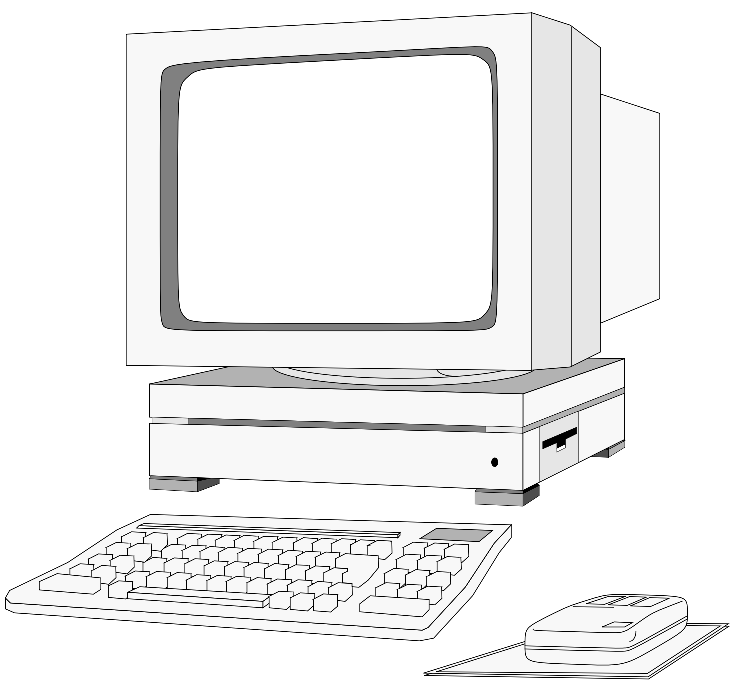
Systems

- **What's in a system?**
  - **IPC - Interprocess Communication**
  - **Memory - Physical / Virtual**
  - **I/O - Input and Output**
  - **System Calls**
- **What is important for gaming?**
  - **Consistency**
  - **Caching**



Input and Output (I/O)

- **I/O bound processes**

  When a process does a lot of I/O operations, we call it I/O bound.
  - *Latency* is a term for the average amount of time between transactions.

- **Compute-bound processes**

  Generally we want to separate computationally intensive actions from threads which are I/O-bound. Excessive computation between I/O calls increases the latency of a server thread.

- **Waiting for I/O**

  `select()`

  Waits until an I/O operation can be performed on a set of file descriptors.
  - select() can operate on any kind of file descriptor, including pipes, sockets, and normal files.

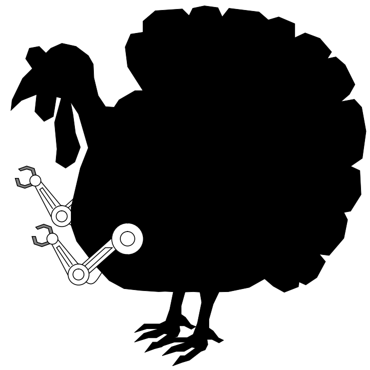
  `XtAddInput()`

  When you want Xt or Motif to inform you that data is available.

- **Reading and writing**

  1. Use read() or fread() for read operations.
     read() is POSIX. fread() is ISO C.
  2. Use write() or fwrite() for write operations.
     write() is POSIX. fwrite() is ISO C.

Robo-Turkey Rex is a C hacker.



- **Misc. commands**

  `ioctl()`

  The C hacker's catch-all.
  - Communicates with the device driver layer of abstraction.

  You'd use this if you wanted to do something like setting your terminal to return one character at a time rather than line-buffer.

  `fcntl()`

  Being specific about how you want to access files.
  - Communicates with the file system layer of abstraction.

  You'd use this if you wanted to make reads and writes non-blocking, ie. your process does not have to wait while the data is written.
  - read() will set *errno* to EAGAIN if you want non-blocking reads and there is nothing available to be read.

- **Where to find out more**

  1. The UNIX Man Pages

     Check out ioctl, termio, fcntl, etc.

Interprocess Communication (IPC)

- **BSD Sockets**

1. What is a socket?

    A socket is an abstraction within the file system which allows for data to be sent between processes (perhaps on different computers).
    - In most cases, sockets are operated on in the same way that you operate on files.

2. How to create a socket

    `socket()`

    Creates a socket. Hard to remember, no?

    `bind()`

    Binds a socket to a particular address or port number.
    - See /etc/services for a list of common port numbers.
    - An example of a port is TCP/IP 80, which is HTTP (ie. what most WWW browsers need to be aware of).

3. How to be a server

    `listen()`

    Tells the OS that you want this socket to be able to accept connections. You specify how many connections can be queued at once.

    `accept()`

    Accepts a connection from another process on the given socket. If the socket is non-blocking, it will return an error indicating that there are no pending connections.

4. How to be a client

    `connect()`

    Connects your socket to another address.

5. Connectionless (IP/UDP ports)

   UDP allows for connectionless packet-based communication.
   - *Advantage*: UDP is *fast*.
   - *Disadvantage*: UDP is unreliable with regards to packet loss.

6. Connection-oriented (TCP/IP streams)

   TCP streams provide something more familiar to most people, and that is a sequenced flow of data (much like a file).
   - With a STREAM based socket, file and network I/O are both abstracted by the file system.
   - *Advantage*: TCP is reliable and operates just like a file.
   - *Disadvantage*: TCP is not as fast as UDP.
   I think that time comparisons were made in the Stevens book (see below for the reference).

- **Pipes**

1. Pipes are *fast*!

   Pipes are blazingly fast, because the protocol which they use is amazingly simple.

2. Only on a single machine

   You can't use a pipe to communicate between to machines on a network.

3. UNIX loves pipes

   Exaggeration: Most everything is done with pipes.
   Reality: Its hard to get by without them.
   - "grep HELLO_THERE | more"

- **UNIX domain sockets**

  UNIX domain sockets are defined only within the context of one machine.
  - UNIX domain sockets can not be used for inter-machine communication.
  - *Pipes* are defined in terms of UNIX domain sockets in the BSD operating system.

- **TLI - System V IPC *(Don't use it!)***

  1. Not as portable as sockets.

     Most operating systems, even Windoze and MacOS, have BSD-style socket libraries.

- **Remote Procedure Calls vs. Message Passing**

  1. Remote Procedure Calls (RPC)

     RPC mimics a function call interface. An interface is specified by the programmer, typically in an Interface Definition Language (IDL) and compiled by a special program (IDL compiler).
  2. Sun RPC

     - Synchronous
     - Asynchronous with threads
  3. DCE RPC

     - Heinous
  4. Message Passing

     - Message passing is simply the sending of data across the network in some defined form.
     - Message passing can be used to simulate RPC.

- **Where to find out more**

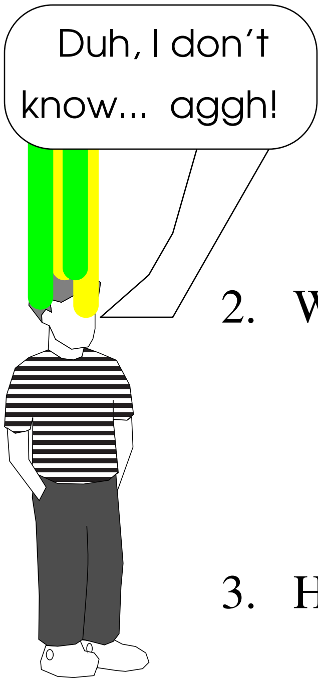
1. UNIX Network Programming

   by W. Richard Stevens
   - This is *the* book. Apparently Stevens intended a second edition of this book, but he couldn't think of anything that he could change.

2. The UNIX Man Pages

   Section 3n is the networking section on Solaris.
   - Try "man -s 3n socket" from a shell.

System Calls vs. Library Calls



1. They are two different things.

   It is said that this is one of the most commonly asked questions at a job interview:
   "What is the difference between a system call and a library call?"
   - A system call involves a trap into the kernel.
   - A library call does not.

2. What does that mean to me?

   A system call could be much slower than a library call, because of the time which it takes to context switch into kernel mode.
   - This might result in a flushing of the TLB -- *Cache misses*!

3. How can I avoid system calls?

   - Write more data at once (see the caching section). write() is a system call.

- **Where to find out more**

1. The UNIX Man Pages

   - Section 2 of the man pages contains system calls.
   - Section 3 of the man pages contains library calls.
   If you don't know what function foo() is, try "man -F -a foo".... It'll tell you.

2. Adv. Programming in the UNIX Env.

   by W. Richard Stevens
   - Stevens is a UNIX wizard.
   This book documents a lot of system calls, and goes as far as to say whether they are BSD, SysV, or POSIX.

Memory

- **Virtual Memory**

  1. Memory which looks like real memory

     ... but is really on disk. Sneaky, sneaky.

  2. Beware of paging and swapping

     Paging occurs when a single piece of memory (or *page*) is either sent to disk (if not used for a while) or taken from disk (if just accessed).
     Swapping occurs when a process has been sleeping for a while and the operating system writes the entire process image to disk.
     - If your game client is swapped to disk or pages a lot, things will slow down immensely.
     - Some operating systems support memory locking, but usually you have to be the super-user.

- **Memory Management**

  1. Inferior Operating Systems

     Most people feel that they have to do their own memory management in operating systems like DOS.
     - I'll cover this if there is time.

  2. Saving time by preallocating

     On UNIX systems, where a memory allocation can result in a system call and a context switch, people might want to pre-allocate data structures which they believe will be in frequent use.
     - A missile object is such a data structure. In the context of a video game, you might want to create a bunch of missiles in advance and simply take them from a list as necessary.

Caching

- **Minimizing I/O**

1. Read as much as possible at once!

   A process which reads 10 bytes at a time will run orders of magnitudes *slower* than a process which reads 1k or more.

2. Write as much as possible at once!

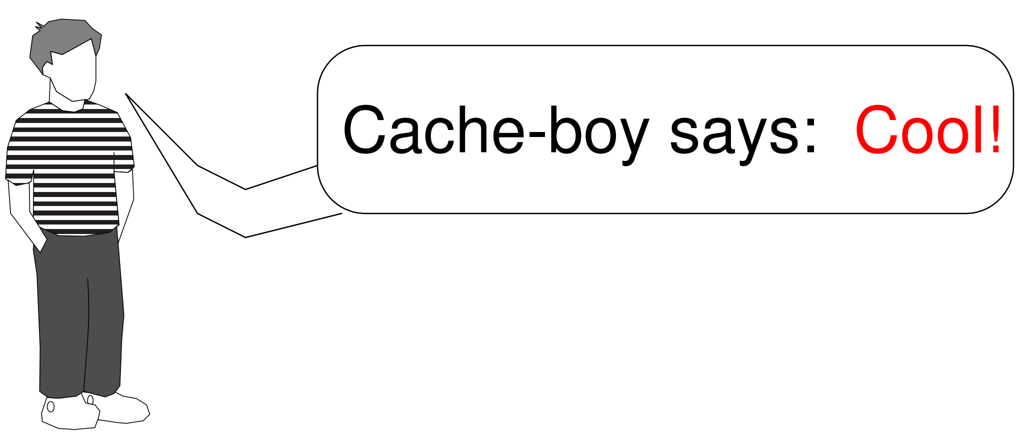
   The same can be said about writing.
   - The file descriptor which you are writing to might be a socket connection to another machine.

3. Polling is good, but callbacks are better.

   When you poll, you are *giving away* valuable cycles of computation time.
   - More about this on Thursday.

- **Save everything which doesn't change**

  If you keep a copy of your world representation on all of the machines that your game is running on, then you only have to send changes (which on the average will be much smaller).



Portability

- **POSIX and ISO C**

  POSIX is the IEEE standard for UNIX systems. ISO C is the international standard for the C language and associated libraries. It is a superset of ANSI.
  - Don't confuse the International *Standards* Organization with the International *Socialist* Organization.

  1. ISO C Library calls

     `memset()`

     Better than bzero().

     `memcpy()`

     Better than bcopy().

     `... you see what I'm getting at...`

- **X11/Xt/Motif**

  1. X Windows *is* portable.

     At least between those operating systems which have functional X servers.

- **Non-POSIX**

  1. select() is not in POSIX

     Doh. Most, if not all, POSIX operating systems support it.

  2. socket() is not in POSIX

     God, what use is POSIX, anyway?
     - Again, most POSIX operating systems support the BSD socket interface.

- **Byte Ordering**

1. What is byte-order?

   Different machine architectures have different representations of numbers. Byte-order deals with the ordering of bytes within a machine word.
   - There are two common types of byte-order:
   Big-endian: Most significant byte first.
   Little-endian: Least significant byte first.

2. What byte-order does my machine use?

   Big-Endian: Most RISC-based workstations, including the SPARC (Sun), MIPS (SGI), Alpha (DEC), etc. Also the Motorola processors, PowerPC and 68000 (including Macintosh).
   Little-Endian: Most (*heh*!) CISC based processors... *Intel* and umm... Intel. Oh, yeah, the VAX used it too.

3. What's the problem?

   How do you send a binary value across the network and have it be understood by machines of different architectures?

4. Network Byte Order

   The standard set by the designers of the Internet protocol families (TCP/IP) that big-endian byte-order be used.
   - This standard was also adopted by other network architectures.

5. How do I solve the problem?

   `htonl(), ntohl(), htons(), ntohs()`

   These functions convert to and from network byte-order. (Think of them as host-to-network and network-to-host functions).
   - Provided by BSD compatible socket libraries.
   - They are NULL macros on machines which use network byte-order.

CS169 (Blatant Plug)

1. Its a lot of fun.
   Learn how to fork().
2. Its a hard course.
   Contrary to popular opinion, it will be getting harder.
3. You'll learn a lot.
   Audit if you must.
4. Meet Tom Doeppner and Tim Brennan.
   They're neat.